Testing

Contents

[Test Plan 1](#_Toc379732398)

[Test data 2](#_Toc379732399)

[Utilities.js unit tests 3](#_Toc379732400)

[utilities.js unit tests – Evidence 7](#_Toc379732401)

[9](#_Toc379732402)

[10](#_Toc379732403)

[utilities.js unit tests – Failed tests 11](#_Toc379732404)

[Ant.js – unit tests 12](#_Toc379732405)

[Ant.js unit tests – Failed tests 14](#_Toc379732406)

[*Test* ***16*** 14](#_Toc379732407)

[Nest.js tests 15](#_Toc379732408)

[Worker.js tests 15](#_Toc379732409)

[Soldier.js tests 16](#_Toc379732410)

[Compatibility tests 17](#_Toc379732411)

[Stress tests 22](#_Toc379732412)

[User interface functional testing 25](#_Toc379732413)

[Biological tests 31](#_Toc379732414)

[Pheromones 31](#_Toc379732415)

[Searching for food 31](#_Toc379732416)

[Trial patterns 32](#_Toc379732417)

# Test Plan

* Black box unit testing (test driven development)
* White box testing
* Integration tests
* State that erroneous data does not occur due to sliders
* Start that extreme data does not occur due to being a simulation

Utilities is unit tested as it contains functions which perform a function on the input i.e. more mathematical.

Ant is unit tested as it is the basis of the worker, queen and soldier classes.

The rest of the code is not unit tested as it contains mostly logic.

*Note*: evidence for tests is found at the test id (where is test code?)

*Note*: Omitted screen shots to reduce document size.

*Note*: Evidence for many tests are written coded test cases and console outputs.

# Test data

## Utilities.js unit tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test | Function | Tests | Reason | Input | Expected | Result |
| 1 | validateDirection | Returns an angle in the range 0 ≤ θ < 2π | **Typical** | 5π | 1π | **PASS** |
| 2 | **Typical** | 0 | 0 | **PASS** |
| 3 | **Typical -** Negative value | -2π | 0 | **PASS** |
| 4 | angleTo | Returns the correct angle to get from coord to target | **Typical** | coord : {x : 5, y : 5}, target : {x : 7, y : 5} | ½ π | **PASS** |
| 5 | **Typical** | coord : {x : 5, y : 5}, target : {x : 7, y : 7} | 3π/4 | **PASS** |
| 6 | **Boundary –** Coordinate on top of ant i.e. navigating to its own position | coord : {x : 5, y : 5}, target : {x : 5, y : 5} | π/2 | **PASS** |
| 7 | **Typical -** wrapped direction i.e. going off map to get to position | {x : 8, y : 8}, target : {x : 1, y : 1} | 3π/4 | **FAIL** - revised and passed with function update |
| 8 | **Typical -** wrapped direction | coord : {x : 7, y : 8}, target : {x : 3, y : 2} | ≈ 3.38657 | **PASS** |
| 9 | boundary | Returns the correct warped coordinate | **Typical** | {x : 4, y : 2} | {x : 4, y : 2} | **PASS** |
| 10 | **Boundary -** Both x and y boundaries | {x : 10, y : 10} | {x : 0, y : 0} | **PASS** |
| 11 | **Typical -** X-axis boundary only | {x : 10, y : 4} | {x : 0, y : 4} | **PASS** |
| 12 | **Typical -** Y-axis boundary only | {x : 7, y : 10} | {x : 7, y : 0} | **PASS** |
| 13 | clone | Produces an exact clone of an object | **Typical** | {a : [1, 2, 3], b : {c : 4, d : '5'}, e : {f : {g : 6}}, h : [['end', 'of', ['object']]]} | {a : [1, 2, 3], b : {c : 4, d : '5'}, e : {f : {g : 6}}, h : [['end', 'of', ['object']]]} | **PASS** |
| 14 | Produces a copy of the object i.e. not by reference | **Typical** | let b = {a : 1} | {a : 1} != b | **PASS** |
| 15 | coordToIndex | Returns the correct index for a specific coordinate | **Typical** | {x : 0, y : 0} | 0 | **PASS** |
| 16 | **Typical** | {x : 5, y : 2} | 25 | **PASS** |
| 17 | distance | Returns the correct distance | **Boundary -** Both the wrapped distance and the regular distance are the same | coord1 : {x : 0, y : 0}, coord2 : {x : 0, y : 0} | 0 | **PASS** |
| 18 | **Typical** | coord1 : {x : -2, y : 1}, coord2 : {x : 1, y : 5} | 5 | **PASS** |
| 19 | getBlock | Returns a block of cells the correct size | **Boundary -** 0 size | coord : {x : 5, y : 5}, size : {width : 0, height : 0} | A single block centred at {x : 5, y : 5} | **PASS** |
| 20 | **Typical** | coord : {x : 5, y : 5}, size : {width : 3, height : 3} | A 7 wide and 7 tall block centred at {x : 5, y : 5} | **PASS** |
| 21 | **Typical -** Non square | coord : {x : 5, y : 5}, size : {width : 1, height : 3} | A 3 wide and 7 tall block centred at {x : 5, y : 5} | **PASS** |
| 22 | Returns a block of cells the correct position | **Typical** | coord : {x : 1, y : 3}, size : {width : 2, height : 2} | A 5 wide and 5 tall block centred at {x : 1, y : 3} which wraps to the other side of the map | **PASS** |
| 23 | **Boundary -** Wrapped in all quadrants | coord : {x : 0, y : 0}, size : {width : 1, height : 1} | A 3 wide and 3 tall block centred at {x : 0, y : 0} which wraps to every corner of the map | **PASS** |
| 24 | getCellCoord | Returns the correct cell for a particular coordinate | **Typical** | {x : 7.2, y : 3.6} | {x : 7, y : 3} | **FAILED -** revised and passed with function update |
| 25 | getSector | Returns sector of correct radius | **Typical** | coord : {x : 25, y : 25}, radius : 15, direction : 0, angle : 2π | A circle of radius 15 centred at {x : 25, y : 25} | **PASS** |
| 26 | **Typical** | coord : {x : 25, y : 25}, radius : 6, direction : 0, angle : 2π | A circle of radius 6 centred at {x : 25, y : 25} | **PASS** |
| 27 | Returns sector at the correct angle | **Typical** | coord : {x : 25, y : 25}, radius : 15, direction : 0, angle : π | A sector of radius 15 centred at {x : 25, y : 25} with an angle of π i.e. a semi-circle | **PASS** |
| 28 | **Typical** | coord : {x : 25, y : 25}, radius : 15, direction : 0, angle : π/4 | A sector of radius 15 centred at {x : 25, y : 25} with an angle of π /4 | **PASS** |
| 29 | Returns sector in correct direction | **Typical** | coord : {x : 25, y : 25}, radius : 15, direction : π, angle : π /4 | A sector of radius 15 centred at {x : 25, y : 25} with an angle of π/4 pointing downwards | **PASS** |
| 30 | **Typical** | coord : {x : 25, y : 25}, radius : 15, direction : 6π/ 4, angle : π /4 | A sector of radius 15 centred at {x : 25, y : 25} with an angle of π/4 pointing along the centre line of the 4th and 3rd quadrants | **PASS** |
| 31 | **Typical** | coord : {x : 25, y : 25}, radius : 15, direction : 3.657, angle : π/4 | A sector of radius 15 centred at {x : 25, y : 25} with an angle of π/4 pointing along the centre line of 3.657 radians | **PASS** |
| 32 | indexToCoord | Returns the correct coordinate from a specific index | **Typical** | 25 | {x : 5, y : 2} | **PASS** |
| 33 | **Typical** | 0 | {x : 0, y : 0} | **PASS** |
| 34 | randColour | Generates random colours (run 3 times) | **Typical** | N/A | A HEX colour | **PASS** |
| 35 | randFloat | Returns a value within a specific range | **Typical** | {min : -1, max : 1} | In range -1 to 1 inclusive | **PASS** |
| 36 | **Typical** | {min : -0.1, max : 0.1} | in range -0.1 to 0.1 inclusive | **PASS** |
| 37 | **Typical** | {min : 0, max : 2} | In range 0 to 2 inclusive | **PASS** |
| 38 | randInt | Returns a value within a specific range | **Typical** | {min : 0, max : 100} | In range 0 to 100 inclusive | **PASS** |
| 39 | **Boundary -** No range of values | {min : 0, max : 0} | 0 | **PASS** |
| 40 | **Typical** | {min : -20, max : 20} | In range -20 to 20 inclusive | **PASS** |
| 41 | randProperty | Returns a random property from an object literal (ran 3 times to show not the same each time) | **Typical** | {a : 0, b : 1, c : 2} | either 'a', 'b' or 'c' (random each time) | **PASS** |
| 42 | scaleCoord | Scales coordinates correctly | **Typical** | {x : 4, y : 2} | {x : 23, y : 12} | **PASS** |

### utilities.js unit tests – Evidence

### 

Test 28

Test 25

Test 31

Test 30

Test 29

Test 27

Test 26

### utilities.js unit tests – Failed tests

#### Test **7**

Test **7** failed as angleTo did not pick the angle which would lead to the shortest path to the target coordinate. The correct angle would have pointed of the map, as the wrapped distance was shorter than the normal distance. This was because angleTo did not take wrapped directions into account in its original form:

**function** angleTo(coord, target) {

**var** dx = target.x - coord.x;

**var** dy = target.y - coord.y;

**return** Math.atan2(dy, dx) + Math.PI/2;

}

The new function takes into account wrapping around the map to get the angle of the shortest path. It uses a similar algorithm to the one in the distance function.

**function** angleTo(coord, target) {

**if** (GRID\_SIZE.width **-** Math.abs(target.x **-** coord.x) **>** Math.abs(target.x **-** coord.x)) {

        dx **=** target.x **-** coord.x;

    } **else** {

        dx **=** GRID\_SIZE.width **-** (target.x **-** coord.x);

    }

**if** (GRID\_SIZE.height **-** Math.abs(target.y **-** coord.y) **>** Math.abs(target.y **-** coord.y)) {

        dy **=** target.y **-** coord.y;

    } **else** {

        dy **=** GRID\_SIZE.height **-** (target.y **-** coord.y);

    }

**return** Math.atan2(dy, dx) **+** Math.PI **/** 2;

}

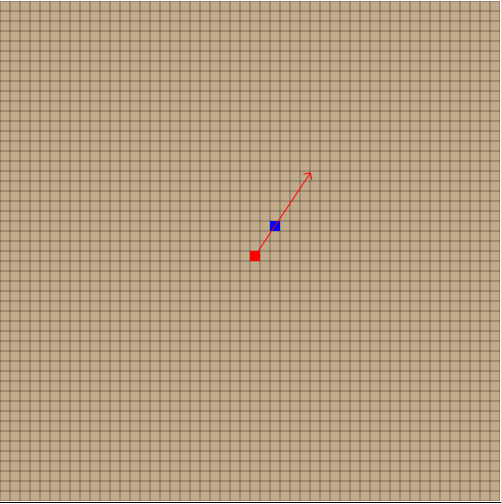
#### Test **24**

Test **24** originally failed due to the getCellCoord function using Math.round rather than Math.floor in its operations, this meant that if the fractional part of x or y was 0.5 or greater the function would be out by one cell.

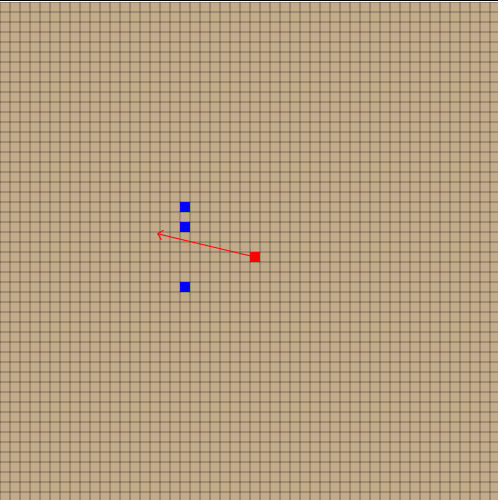
## Ant.js – unit tests

| Test | Function | Tests | Reason | Input | Expected | Result |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Ant.scan | Adds all ants within viewing distance to ant.itemsInView.ants | **Typical** | No ants in view | this.itemsInView.ants.length = 0 i.e. cannot see any ants in view | **PASS** |
| 2 | **Typical** | Multiple ants within viewing distance | this.itemsInView.ants.length = 4 | **PASS** |
| 3 | Adds food in viewing distance to ant.itemsInView.food | **Typical** | No food in view | this.itemsInView.food.length = 0 i.e. cannot see any food in view | **PASS** |
| 4 | **Typical** | Multiple pieces of food in view | this.itemsInView.food.length = 3 | **PASS** |
| 5 | Ant.secrete | Adds pheromone of correct concentration | **Typical** | No pheromones to start | A new pheromones of concentration 0.5 | **PASS** |
| 6 | **Typical -** Adding pheromone | A pheromone of the same species | A new pheromones of concentration 0.9 i.e. 0.4 + 0.5 | **PASS** |
| 7 | **Typical -** Not adding pheromones as different species | A pheromone of a different species | A new pheromones of concentration 0.5 | **PASS** |
| 8 | **Typical -** Doesn’t exceed MAX\_PHEROMONE\_CONCENRATION | A pheromone of the same species | A new pheromones of concentration 1 | **PASS** |
| 9 | Ant.atNest | Ant is on nest or not | **Typical** | not on the nest | FALSE | **PASS** |
| 10 | **Typical** | On the nest | TRUE | **PASS** |
| 11 | Ant.seeNest | Can see nest when in range | **Typical** | Cannot see the nest | FALSE | **PASS** |
| 12 | **Typical** | Nest within view | TRUE | **PASS** |
| 13 | Ant.smell | Adds all pheromones within range to pheromonesInRange | **Typical** | No pheromones in rang | this.pheromonesInRange.length = 0 i.e. cannot smell any pheromones in view | **PASS** |
| 14 | **Typical -** Shouldn't read pheromone it’s on | 4 Pheromones in range with one on top of the ant | this.pheromonesInRange.length = 3 | **PASS** |
| 15 | Ant.takeFood | Takes correct amount of food | **Typical** | void(0) i.e. No food | 0 | **PASS** |
| 16 | **Typical** | A single piece of food of amount 3 | After three runs of Ant.takeFood food = void(0) | **FAIL -** PASSED after re writing function |
| 17 | Ant.wonder | Ant picks correct direction given some amount of pheromones | **Typical** | No pheromones in view | this.direction === this.prioritizeDirection | **PASS** |
| 18 | **Typical** | A single pheromone | Ant is pointing at the pheromone | **PASS** |
| 19 | **Typical** | Multiple pheromones | Ant is pointing slightly upwards to the right | **PASS** |

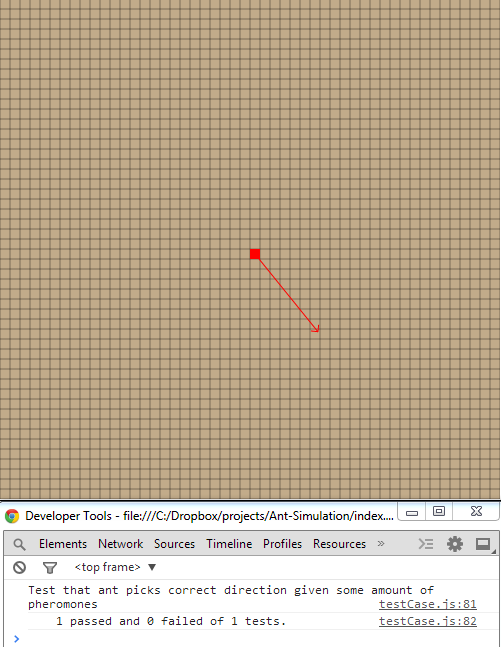
Ant.js unit test – Evidence



Test 18



Test 19



Test 17

### Ant.js unit tests – Failed tests

#### Test **16**

Test 16 failed originally due to an error in the Ant.takeFood function.

**if** (**this**.isFood(food))

food.removeFromMap();

There was a missing not in the stamen which meant that the food would be removed after one piece of food was taken. Here is the fixed version:

**if** (**!this**.isFood(food))

food.removeFromMap();

## Nest.js tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test | Function | Tests | Reason | Input | Expected | Result |
| 1 | Nest.reproduce | Produces the correct probabilities | **Typical** | Soldier probability = 1, worker and queen probabilities = 0. Run 10,000 times to check probabilities. | Only soldier ants produced. | **PASS** |
| 2 | **Typical** | Soldier, Worker and Queen probabilities are all 1. Run 10,000 times to check probabilities. | A 1/3 chance of each type of ant | **PASS** |
| 3 | **Boundary** – sum will be 0 therefore dividing by 0 should produce null. | Soldier, Worker and Queen probabilities are all 0. Run 10,000 times to check probabilities. | No ants created | **PASS** |

## Worker.js tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test | Action | Tests | Reason | Input | Expected | Result |
| 1 | Searching for food. | Tests an ant will search for food according to the algorithm X | **Typical** | A single piece of food on the map. | The ant to wonder around, scanning its surroundings and interpreting the items in view for food targets. When it spots food, its goal updates to getting food. | **PASS** |
| 2 | Retrieving food. | Tests an ant will retrieve food which it has targeted correctly. | **Typical** | An ant facing a single piece of food which is in viewing distance. | The ant will move towards the food until it is on top of it. The ant will determine how to best use the food i.e. eat or carry. The ant will carry out the action associated with its choice. A single piece of food should be removed from the food source and either added to the ants carrying amount or added to its health. | **PASS** |
| 3 | Dropping food. | Tests an ant will drop food to its own nest. | **Typical** | An ant carrying 3 pieces of food and a nest of the same species. | The ant to wonder around until it sees the nest. Then move directly towards the nest. Upon reaching the nest, spend 3 ticks until all food is gone. The ant’s goal will then update to finding food. All the while pheromones are being laid. | **PASS** |

## Soldier.js tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test | Action | Tests | Reason | Input | Expected | Result |
| 1 | Guarding its nest. | Tests an ant will guard its nests correctly. | **Typical** | A single soldier ant and its nest. | The ant will search for its nest, when it finds its nest it will move certain distance away, stop and start turning around. | **PASS** |
| 2 | Guarding food. | Tests an ant will guard food correctly | **Typical** | A single soldier ant and a single source of food. | The ant will search for the food. When the ant sees the food, it will stop and start turning around. | **PASS** |
| 3 | Attacking other ants | Tests an ant will attack ants it has targeted correctly. | **Typical** | A single soldier ant and a single worker ant of different species. | [For solider ant only]The ant will wonder around, when it sees the other ant it will target the ant. When the ant is targeted it will begin to follow the ant until it is range to attack. The ant will then attack the other ant reducing its health until death. This will continue to happen until the other ant is dead. | **PASS** |
| 4 | Soldier.soldiersInView | Test that soldier in view returns true when there are soldiers in view | **Typical** | No ants in view | returns false i.e. no soldiers in view | **PASS** |
|  | **Typical** | Multiple ants within viewing distance which are both workers and soldiers | returns true | **PASS** |
|  | **Typical** | A single ant within viewing distance which is not a soldier | returns false | **PASS** |
|  | **Typical** | A single ant within viewing distance which is a soldier | returns true | **PASS** |
| 5 | Test that soldier in view correctly determines if there are friendly (i.e. of same species) ants in view | **Typical** | A solider ant of the same species | returns true | **PASS** |
| 6 | **Typical** | A soldier ant of a different species | returns false | **PASS** |

## Compatibility tests

The tool has been tested on multiple web browsers on multiple operating systems to check its compatibility. This is necessary as the tool may be set for homework and the pupils need to be able to use it at home.

**Note: The canvas HTML 5 element which is used to display the simulation is the same in all browsers which support it for its main features. The use of the canvas element in the simulation is specifically designed to only use the core features of the canvas so that support can be guaranteed by browsers which have an implementation of the canvas element.**

| Test | Description | Reason | Evidence | Result |
| --- | --- | --- | --- | --- |
| 1 | Chrome – Windows 8 | Chrome being the most popular browser and windows 8 being a popular operating systems. | C:\Dropbox\projects\Ant-Simulation\app\tests\Browser compatibility\win8_chrome_27.0.png | **PASS** |
| 2 | Firefox – Windows 8 | Firefox is also a popular web browser and windows 8 being a popular operating systems. | C:\Dropbox\projects\Ant-Simulation\app\tests\Browser compatibility\win8_firefox_20.0.png | **PASS –** The slider input html element is not supported in Firefox and so the inputs are text boxes rather than sliders. |
| 3 | Safari – IOS (iPad) | Safari is the default browser on IOS and so properly most used. The iPad is one of the only portable devices which the simulation can run on smoothly, phones are not powerful enough. | C:\Dropbox\projects\Ant-Simulation\app\tests\Browser compatibility\ios_iPad-3rd_5.1_portrait.jpg | **PASS** |
| 4 | Chrome - Ubuntu | Chrome being the most popular browser and Ubuntu being the most popular Linux distribution this test is likely to catch the small number of people who use a Linux OS. | C:\Users\0x\Downloads\140203-162615-begly.github.io-9871292\140203-165738-chrome-32.0.1700.102-ubuntu-12.04-lts-41d660b1228e545dbe8d57c4dfd6b8d1.png | **PASS** |
| 5 | Safari – OS X | Safari is the default browser on OS X and therefore probably one of the most popular. | C:\Users\0x\Downloads\140203-162615-begly.github.io-9871292\140203-165115-safari-5.1.7-mac-os-x-10.7-8c5186b2a1e265bb6967357aab5963fc.png | **PASS** |

## Stress tests

The simulation went through a series of stress tests to see how well it copes with large numbers of ants. Ants where picked as the independent variable as the number of ants is the primary contribution to the time complexity of the algorithm. Only worker ants are used as they act as a good average complexity (as queen ants require less processing then worker ants and soldier ants require more processing then worker ants).

#### Setup

The simulation was set with a GRID\_SIZE of 250x250. The simulation was slightly modified so that an exact number of ants would be created at random positions at the start of the simulation rather than having to wait for multiple nests to produce enough ants.

The simulation was run on chrome 32 on the windows 7 operating system. The hardware was a school computer similar to the ones used in the biology department and thus a good representation of the performance to be expected when the simulation is used.

#### Frame rate and usability

The frame rate in the simulation will not be noticeable until it gets extremely low e.g. < 2 fps. This is because the frame rate is the same as the tick time, thus meaning higher frame rates will mean the simulation runs faster and lower frame rates means the simulation runs slower. This is done to increase performance of the simulation so that it can run as fast as possible so the user can see how the change of characteristics affect the ant’s behaviour faster.

#### Results

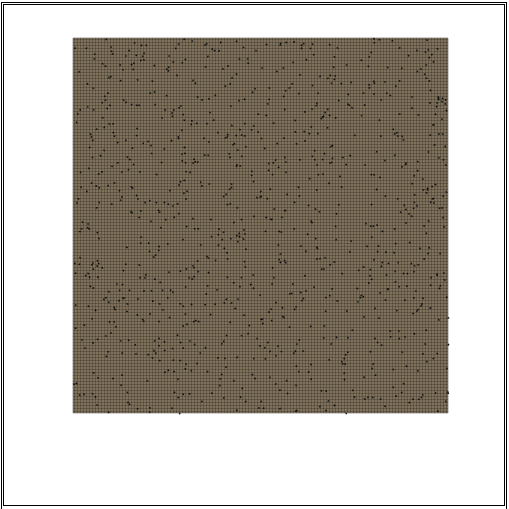
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Tests | Reason | Expected | Result |
| 1 | 1 ant | The starting number of ants on the simulation. This is certain to happen in every simulation. | Very smooth frame rate (> 60fps) | **PASS** |
| 2 | 10 ants | A small nest of ants (< 20 ants per nest). This will very nearly happen every simulation run. | Very smooth frame rate (> 60fps) | **PASS** |
| 3 | 100 ants | Multiple medially sized nests (> 20 ants per nest). This will happen in simulations left on few about 2 minutes | Smooth frame rate (> 30fps) | **PASS** |
| 4 | 1000 ants | Multiple large nests (> 80 ants per nest). This is not likely to happen to most users. | Useable frame rate (> 5fps) | **PASS** |
| 5 | 10000 ants | Multiple very large nests (> 200 ants per nest). This may not happen atoll in most simulations, no matter how long they are left on for due to the characteristics chosen and either ants dying out due to being out competed for food or lack of food. | Very choppy frame rate (> 0.25fps) | **PASS** |
| 6 | 100000 ants | Many multiples of extremely large colonies of ants (> 5000 ants per nest). This is extremely unlikely to occur as it would require a significant amount of time to get the simulation to this point as well as an extremely large map and large food source. | Unusable (< 0.25fps) | **PASS** |

#### C:\Dropbox\projects\Ant-Simulation\app\tests\stress\1 ant.PNGC:\Dropbox\projects\Ant-Simulation\app\tests\stress\10 ants.PNGC:\Dropbox\projects\Ant-Simulation\app\tests\stress\100 ants.PNGStress tests – Evidence

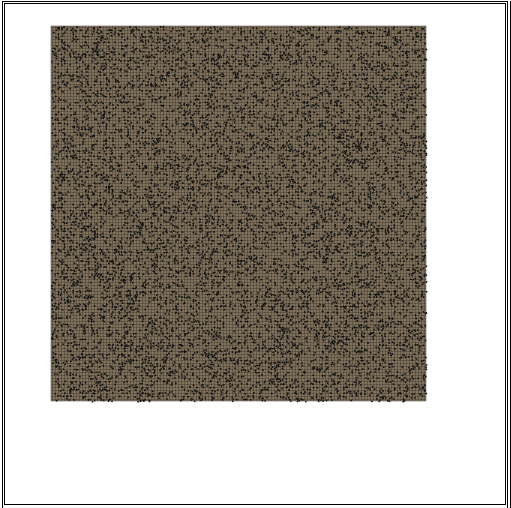
Test - 1 ant (200 fps)

Test - 10 ants (100 fps)

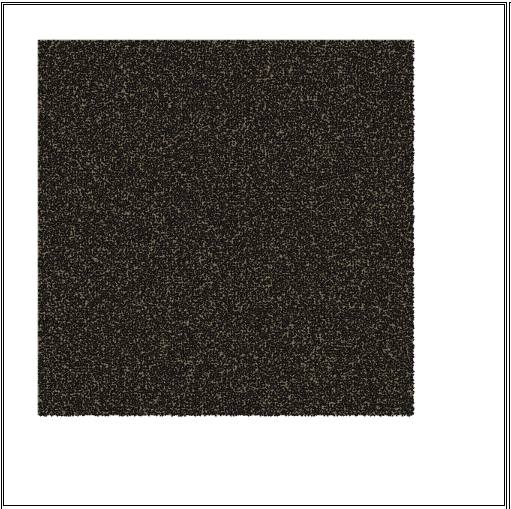
Test - 100 ants (40 fps)



Test - 1000 ants (8 fps)



Test - 10000 ants (1 fps)

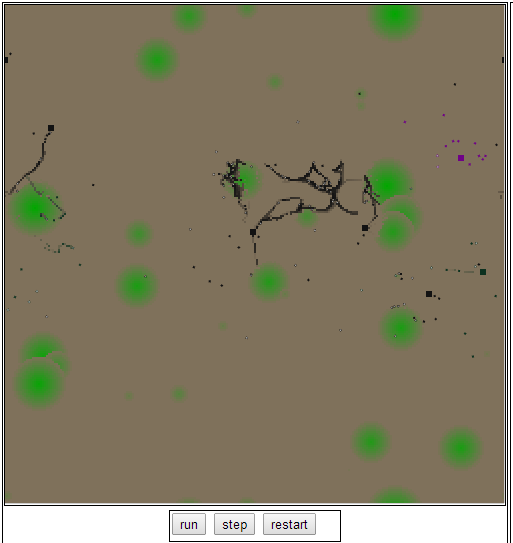


Test - 100000 ants (N/A fps)

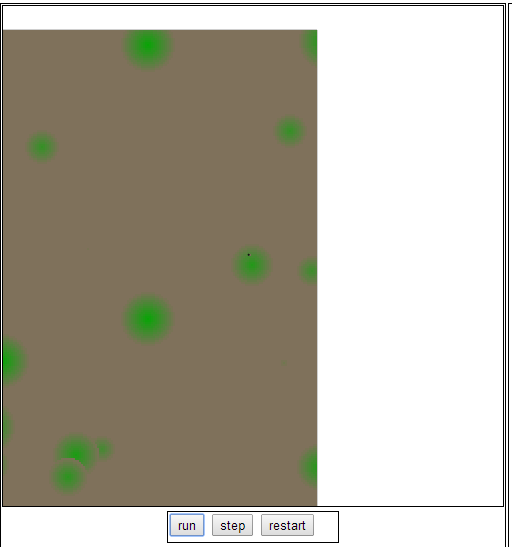
## User interface functional testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Tests | Method | Expected | Result |
| 1 | The run button starts and stops the simulation. | Press the pause button and then press the run button. | When pause button pressed simulation stops and the pause button will change to a run button. When the run button is pressed the simulation starts from the place it left off and the run button changes back into a pause button. | **PASS** |
| 2 | Press the space bar. | **PASS** |
| 3 | The step button steps the simulation forwarded a single tick. | Press the step button. | All ants to go through a single update cycle i.e. do a single action such as pick up a piece of food or attack an ant. All ants (apart from those collecting food) will move an amount equal to their species speed characteristic. | **PASS** |
| 4 | Press the “s” key. | **PASS** |
| 5 | The restart button restarts the simulation. | Press the restart button. | The simulation restarts, keeping all characteristics for the starting species the same. | **PASS** |
| 6 | Press the “r” key. | **PASS** |
| 7 | Species data selection. | Click on the species from the species panel. (Selecting species 46) | Map to centre on the first nest in the spices. The selected species to be highlighted in the species data panel. The characteristics of the selected species to appear in the characterises panel. | **PASS** |
| 8 | Expanding species data | Press the + button of species 87 | Data about the species expands showing the species colour, number of ants, number of nests and amount of food. | **PASS** |
| 9 | Contracting species data | Press the – button of species 87 | Data about the species colour, number of ants, number of nests and amount of food to be hidden. | **PASS** |
| 10 | Pan around simulation | Click and drag mouse on simulation. | Simulation to move under the mouse until mouse is released. | **PASS** |
| 11 | Press arrow keys. | **PASS** |
| 12 | Zoom into simulation | Scroll mouse wheel. | Simulation to either zoom or zoom out depending on direction of scroll/+ or – key. | **PASS** |
| 13 | Press + and – keys. | **PASS** |
| 14 | The random button randomizes the selected species characteristics. | Pressing the random button. | The characteristics of the selected species to be changed to random values. The update button to highlight. And the ants health to change. | **PASS** |
| 15 | The reset button resets selected species characteristics to default values. | Press the reset button. | The characteristics of the selected species to change to the default values. The update button to highlight. And the ants health to change. | **PASS** |
| 16 | The update button pushes the changed value to the simulation. | Press the update button. | The update button to return to its normal colour. The species in the simulation value to be updated. | **PASS** |

Test 5

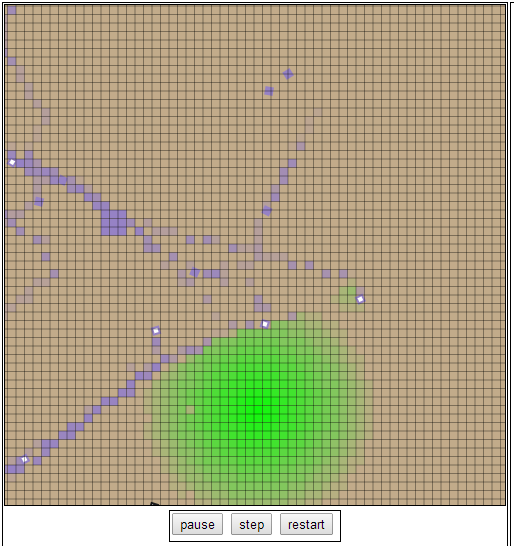
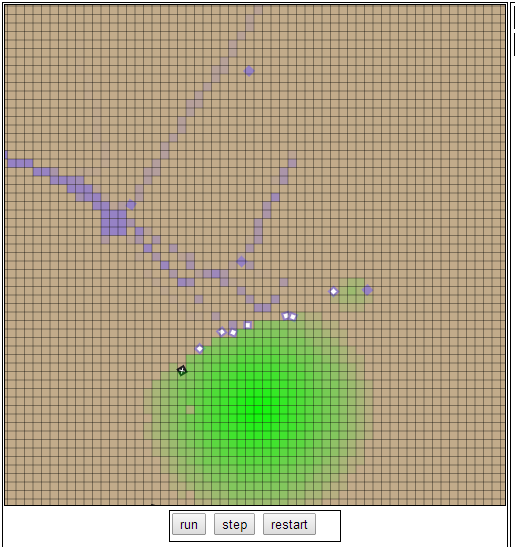


Pre restart



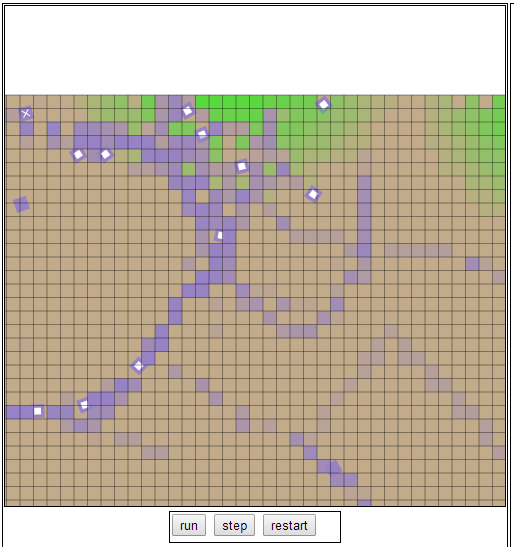
Post restart

Test 1

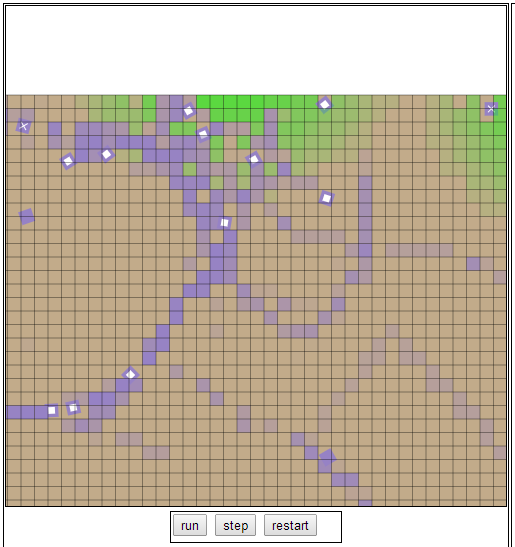


Pause pressed

Run pressed



Pre step



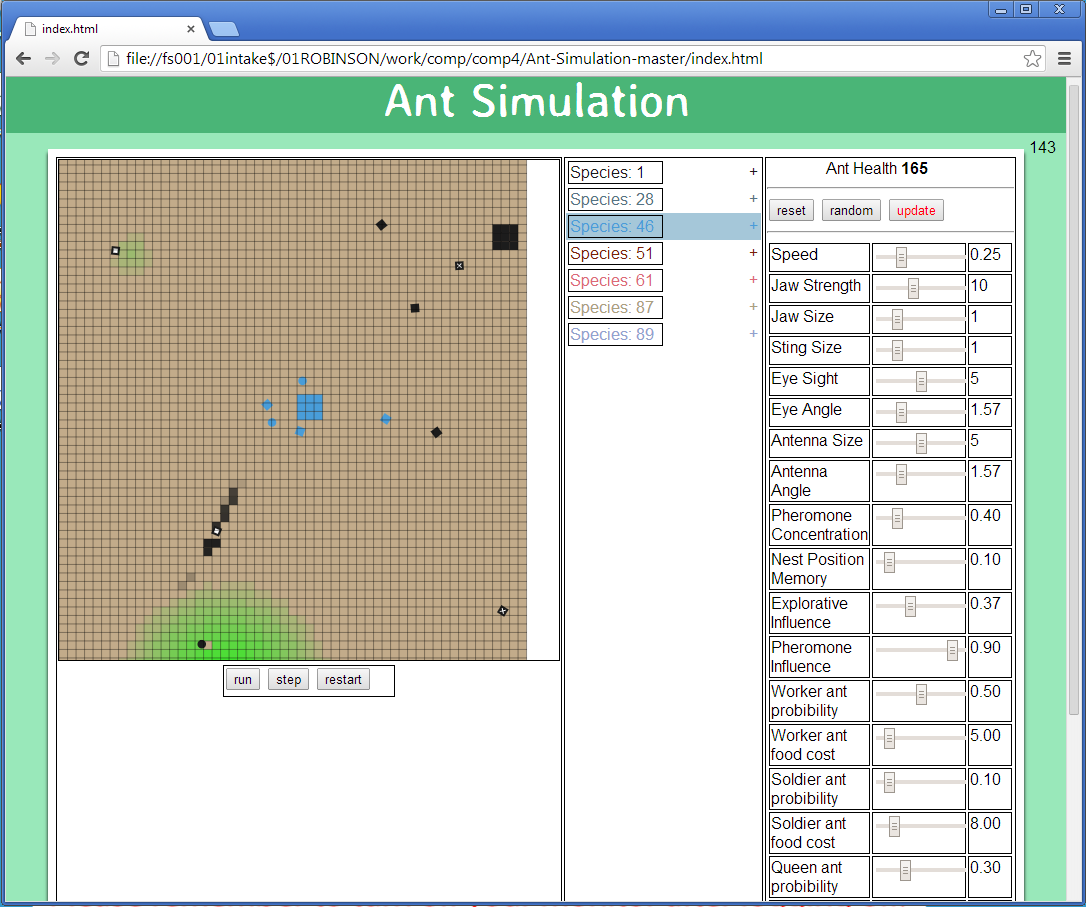
Post step

Test 3

Test

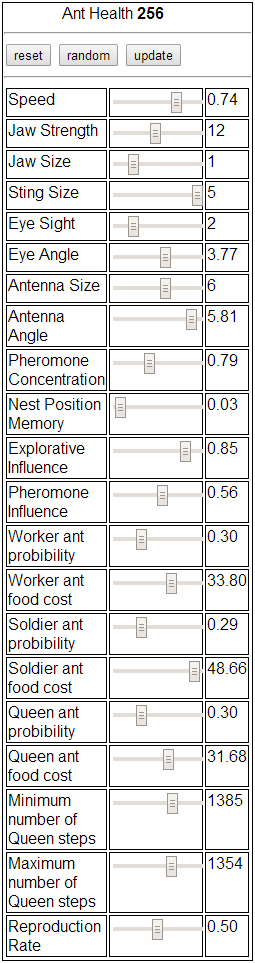


Pre selection



Post selection of species 46

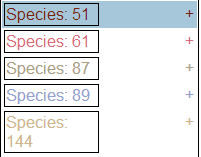
Test 14



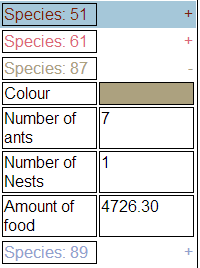
Pre random



Post random



Test 9



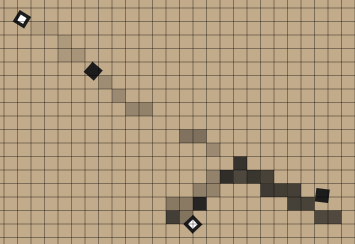
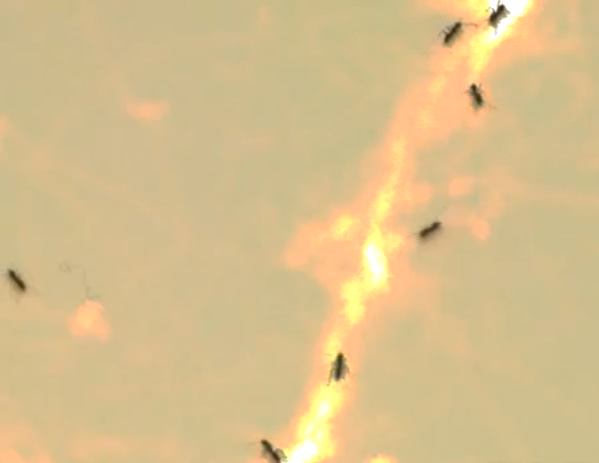
Test 8

## Biological tests

The simulation is based on ants, the behaviour of the simulated ants is changed by altering a species characteristics. This behaviour can sometimes closely resemble real ants however it can be equally a long way of how real ant behave. For example by changing the antenna size characteristic to 0 will mean that ants can no longer smell pheromones. The behaviour of ants in this state will not model the behaviour of real ants (which can smell pheromones). This means that in order to test the biological trueness of the simulation would be to find the characteristics in the simulation which most closely match a real ant, this is very difficult as the simulation only has a limited range of characteristics to choose from while real ants have a much more complex system of genes.

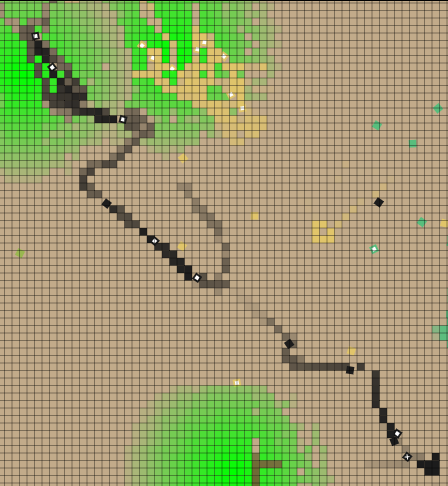
A comparison between real ants and the simulated ants will be used to compare how well the simulated ants model the real ants. Also approximations of behaviour will also be compared between the two.

### Pheromones

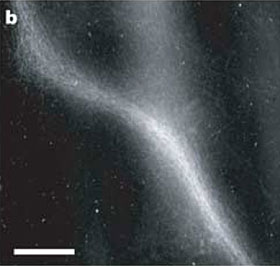
Pheromones laid by real ants are most concentrated where most ants have been as they cumulate. This is seen in the picture of real ants by the brighter yellow colour. This can be seen in the simulation by a darker colour pheromone. In videos, ants will tend to head in the direction which has the strongest concentration of pheromones. Again this is mirrored in the simulation (See tests 17, 18 and 19 in Ant.js unit tests). Real ants will also sometimes leave the trial to go off looking in a new direction. This is shown in the diagram of the simulation where the top right ants did not follow the stronger pheromone trial left by the ant in the centre left. And in the picture of real ants by the ant on the far left of the screen who left the trial in search for food. Finally pheromones in real life evaporate over time, this is also modelled in the simulation as pheromones concentration decrease as time increases.

### Searching for food

Real ants walk randomly until an ant finds food, it then navigates back to the nest (depending on type of ant either following a pheromone, memory or randomly). Other ants then follow the trial left by the first ant to find the food, reinforcing the strength of the trial making it more popular. This can all also be seen in the simulated ants.

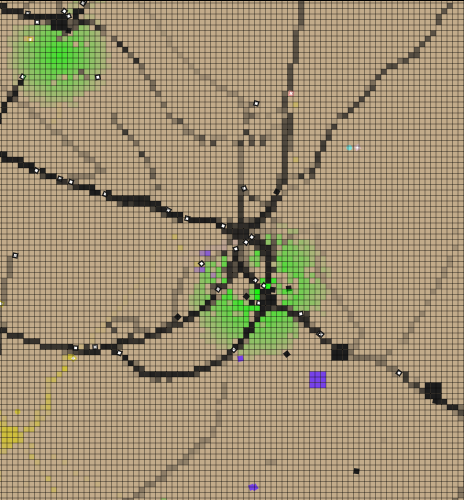
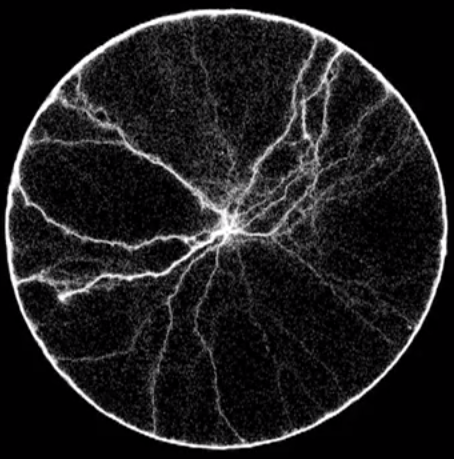
The screen shot shows a trial which leads from the nest to food. A similar looking trial is seen in the picture of real ants leading from the nest to food.

NEST



FOOD

### Trial patterns

Real ant trials come into the nest from all directions, the main trials (strongest concentration pheromones) are roughly 70 degrees apart (to maximise spread to new food sources). This is very similar to the simulation which has trials from all directions and the main trials are separated by about 70 degrees also. The concentration of pheromones is also similar, although the scale of the simulation and the real pheromone trials of ants are different the pattern of most concentrated pheromones around the nest with roughly straight thin lines of high concentration pheromones leading off in multiple directions as well as a small number of considerable weaker pheromones from all directions can be seen in the real pheromones and the simulated.